Bike_Share_Analysis

November 1, 2017

1 2016 US Bike Share Activity Snapshot

1.1 Table of Contents

Introduction

> **Tip**: Quoted sections like this will provide helpful instructions on how to navigate and use a Jupyter notebook.

Over the past decade, bicycle-sharing systems have been growing in number and popularity in cities across the world. Bicycle-sharing systems allow users to rent bicycles for short trips, typically 30 minutes or less. Thanks to the rise in information technologies, it is easy for a user of the system to access a dock within the system to unlock or return bicycles. These technologies also provide a wealth of data that can be used to explore how these bike-sharing systems are used.

In this project, you will perform an exploratory analysis on data provided by Motivate, a bike-share system provider for many major cities in the United States. You will compare the system usage between three large cities: New York City, Chicago, and Washington, DC. You will also see if there are any differences within each system for those users that are registered, regular users and those users that are short-term, casual users.

Posing Questions

Before looking at the bike sharing data, you should start by asking questions you might want to understand about the bike share data. Consider, for example, if you were working for Motivate. What kinds of information would you want to know about in order to make smarter business decisions? If you were a user of the bike-share service, what factors might influence how you would want to use the service?

Question 1: Write at least two questions related to bike sharing that you think could be answered by data.

Answer: 1. Which are the hours that people uses the service mostly? 2. which are the year range of the people who mostly use the service?

> **Tip**: If you double click on this cell, you will see the text change so that all of the formatting is removed. This allows you to edit this block of text. This block of text is written using Markdown, which is a way to format text using headers, links, italics, and many other options using a plain-text syntax. You will also use Markdown later in the Nanodegree program. Use **Shift** + **Enter** or **Shift** + **Return** to run the cell and show its rendered form.

Data Collection and Wrangling

Now it's time to collect and explore our data. In this project, we will focus on the record of individual trips taken in 2016 from our selected cities: New York City, Chicago, and Washington, DC. Each of these cities has a page where we can freely download the trip data.:

- New York City (Citi Bike): Link
- Chicago (Divvy): Link
- Washington, DC (Capital Bikeshare): Link

If you visit these pages, you will notice that each city has a different way of delivering its data. Chicago updates with new data twice a year, Washington DC is quarterly, and New York City is monthly. **However, you do not need to download the data yourself.** The data has already been collected for you in the `/data/` folder of the project files. While the original data for 2016 is spread among multiple files for each city, the files in the `/data/` folder collect all of the trip data for the year into one file per city. Some data wrangling of inconsistencies in timestamp format within each city has already been performed for you. In addition, a random 2% sample of the original data is taken to make the exploration more manageable.

Question 2: However, there is still a lot of data for us to investigate, so it's a good idea to start off by looking at one entry from each of the cities we're going to analyze. Run the first code cell below to load some packages and functions that you'll be using in your analysis. Then, complete the second code cell to print out the first trip recorded from each of the cities (the second line of each data file).

> **Tip**: You can run a code cell like you formatted Markdown cells above by clicking on the cell and using the keyboard shortcut **Shift** + **Enter** or **Shift** + **Return**. Alternatively, a code cell can be executed using the **Play** button in the toolbar after selecting it. While the cell is running, you will see an asterisk in the message to the left of the cell, i.e. `In [*]:`. The asterisk will change into a number to show that execution has completed, e.g. `In [1]`. If there is output, it will show up as `Out [1]:`, with an appropriate number to match the "In" number.

```
In [31]: ## import all necessary packages and functions.
         import csv # read and write csv files
         from datetime import datetime # operations to parse dates
         from pprint import pprint # use to print data structures like dictionaries in
                                   # a nicer way than the base print function.
         import numpy as np
         import pandas as pd
```

```
In [2]: def print_first_point(filename):
            """
            This function prints and returns the first data point (second row) from
```

```
        a csv file that includes a header row.
        """
        # print city name for reference
        city = filename.split('-')[0].split('/')[-1]
        print('\nCity: {}'.format(city))

        with open(filename, 'r') as f_in:
            ## TODO: Use the csv library to set up a DictReader object. ##
            ## see https://docs.python.org/3/library/csv.html           ##
            trip_reader = csv.DictReader(f_in)


            ## TODO: Use a function on the DictReader object to read the     ##
            ## first trip from the data file and store it in a variable.     ##
            ## see https://docs.python.org/3/library/csv.html#reader-objects ##
            first_trip = next(trip_reader)

            ## TODO: Use the pprint library to print the first trip. ##
            ## see https://docs.python.org/3/library/pprint.html     ##
            pprint(first_trip)

        # output city name and first trip for later testing
        return (city, first_trip)

    # list of files for each city
    data_files = ['./data/NYC-CitiBike-2016.csv',
                  './data/Chicago-Divvy-2016.csv',
                  './data/Washington-CapitalBikeshare-2016.csv',]

    # print the first trip from each file, store in dictionary
    example_trips = {}
    for data_file in data_files:
        city, first_trip = print_first_point(data_file)
        example_trips[city] = first_trip


City: NYC
OrderedDict([('tripduration', '839'),
             ('starttime', '1/1/2016 00:09:55'),
             ('stoptime', '1/1/2016 00:23:54'),
             ('start station id', '532'),
             ('start station name', 'S 5 Pl & S 4 St'),
             ('start station latitude', '40.710451'),
             ('start station longitude', '-73.960876'),
             ('end station id', '401'),
             ('end station name', 'Allen St & Rivington St'),
             ('end station latitude', '40.72019576'),
             ('end station longitude', '-73.98997825'),
```

```
             ('bikeid', '17109'),
             ('usertype', 'Customer'),
             ('birth year', ''),
             ('gender', '0')])

City: Chicago
OrderedDict([('trip_id', '9080545'),
             ('starttime', '3/31/2016 23:30'),
             ('stoptime', '3/31/2016 23:46'),
             ('bikeid', '2295'),
             ('tripduration', '926'),
             ('from_station_id', '156'),
             ('from_station_name', 'Clark St & Wellington Ave'),
             ('to_station_id', '166'),
             ('to_station_name', 'Ashland Ave & Wrightwood Ave'),
             ('usertype', 'Subscriber'),
             ('gender', 'Male'),
             ('birthyear', '1990')])

City: Washington
OrderedDict([('Duration (ms)', '427387'),
             ('Start date', '3/31/2016 22:57'),
             ('End date', '3/31/2016 23:04'),
             ('Start station number', '31602'),
             ('Start station', 'Park Rd & Holmead Pl NW'),
             ('End station number', '31207'),
             ('End station', 'Georgia Ave and Fairmont St NW'),
             ('Bike number', 'W20842'),
             ('Member Type', 'Registered')])
```

If everything has been filled out correctly, you should see below the printout of each city name (which has been parsed from the data file name) that the first trip has been parsed in the form of a dictionary. When you set up a `DictReader` object, the first row of the data file is normally interpreted as column names. Every other row in the data file will use those column names as keys, as a dictionary is generated for each row.

This will be useful since we can refer to quantities by an easily-understandable label instead of just a numeric index. For example, if we have a trip stored in the variable `row`, then we would rather get the trip duration from `row['duration']` instead of `row[0]`.

Condensing the Trip Data

It should also be observable from the above printout that each city provides different information. Even where the information is the same, the column names and formats are sometimes different. To make things as simple as possible when we get to the actual exploration, we should trim and clean the data. Cleaning the data makes sure that the data formats across the cities are consistent, while trimming focuses only on the parts of the data we are most interested in to make the exploration easier to work with.

You will generate new data files with five values of interest for each trip: trip duration, starting month, starting hour, day of the week, and user type. Each of these may require additional

wrangling depending on the city:

- **Duration**: This has been given to us in seconds (New York, Chicago) or milliseconds (Washington). A more natural unit of analysis will be if all the trip durations are given in terms of minutes.
- **Month**, **Hour**, **Day of Week**: Ridership volume is likely to change based on the season, time of day, and whether it is a weekday or weekend. Use the start time of the trip to obtain these values. The New York City data includes the seconds in their timestamps, while Washington and Chicago do not. The `datetime` package will be very useful here to make the needed conversions.
- **User Type**: It is possible that users who are subscribed to a bike-share system will have different patterns of use compared to users who only have temporary passes. Washington divides its users into two types: 'Registered' for users with annual, monthly, and other longer-term subscriptions, and 'Casual', for users with 24-hour, 3-day, and other short-term passes. The New York and Chicago data uses 'Subscriber' and 'Customer' for these groups, respectively. For consistency, you will convert the Washington labels to match the other two.

Question 3a: Complete the helper functions in the code cells below to address each of the cleaning tasks described above.

In [3]:
```
def duration_in_mins(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (datum) and
    its origin city (city) and returns the trip duration in units of minutes.

    Remember that Washington is in terms of milliseconds while Chicago and NYC
    are in terms of seconds.

    HINT: The csv module reads in all of the data as strings, including numeric
    values. You will need a function to convert the strings into an appropriate
    numeric type when making your transformations.
    see https://docs.python.org/3/library/functions.html
    """

    miliseconds_to_minutes_conversion_divisor = 1.66667e-5
    seconds_to_minutes_conversion_divisor = 60
    duration = None

    if city == "Washington":
        trip_duration = datum["Duration (ms)"]
        duration = int(trip_duration)
        duration *= miliseconds_to_minutes_conversion_divisor
    else:
        trip_duration = datum["tripduration"]
        duration = int(trip_duration)
        duration /= seconds_to_minutes_conversion_divisor

    return duration
```

```
        # Some tests to check that your code works. There should be no output if all of
        # the assertions pass. The `example_trips` dictionary was obtained from when
        # you printed the first trip from each of the original data files.
        tests = {'NYC': 13.9833,
                 'Chicago': 15.4333,
                 'Washington': 7.1231}

        for city in tests:
            assert abs(duration_in_mins(example_trips[city], city) - tests[city]) < .001
```

In [4]:
```
def time_of_trip(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (datum) and
    its origin city (city) and returns the month, hour, and day of the week in
    which the trip was made.

    Remember that NYC includes seconds, while Washington and Chicago do not.

    HINT: You should use the datetime module to parse the original date
    strings into a format that is useful for extracting the desired information.
    see https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior
    """

    parsed_date = None
    month = None
    hour = None
    day_of_week = None

    start_trip_datum_index = 'starttime'
    datetime_format = '%m/%d/%Y %H:%M'

    if city == 'Washington':
        start_trip_datum_index = 'Start date'

    if city == 'NYC':
        datetime_format += ':%S'

    parsed_date = datetime.strptime(datum[start_trip_datum_index], datetime_format)

    month = parsed_date.month
    hour = parsed_date.hour
    day_of_week = datetime.strftime(parsed_date, '%A')

    return (month, hour, day_of_week)
```

```
        # Some tests to check that your code works. There should be no output if all of
        # the assertions pass. The `example_trips` dictionary was obtained from when
        # you printed the first trip from each of the original data files.
        tests = {'NYC': (1, 0, 'Friday'),
                 'Chicago': (3, 23, 'Thursday'),
                 'Washington': (3, 22, 'Thursday')}

        for city in tests:
            assert time_of_trip(example_trips[city], city) == tests[city]
```

In [5]:
```
        def type_of_user(datum, city):
            """
            Takes as input a dictionary containing info about a single trip (datum) and
            its origin city (city) and returns the type of system user that made the
            trip.

            Remember that Washington has different category names compared to Chicago
            and NYC.
            """

            user_type = None
            user_type_index = 'usertype'

            if city == 'Washington':
                user_type_index = 'Member Type'

            user_type = get_unified_subscription_status(datum[user_type_index])

            return user_type

        def get_unified_subscription_status(user_status):
            if user_status == 'Subscriber' or user_status == 'Registered':
                return 'Subscriber'
            else:
                return 'Customer'


        # Some tests to check that your code works. There should be no output if all of
        # the assertions pass. The `example_trips` dictionary was obtained from when
        # you printed the first trip from each of the original data files.
        tests = {'NYC': 'Customer',
                 'Chicago': 'Subscriber',
                 'Washington': 'Subscriber'}

        for city in tests:
            assert type_of_user(example_trips[city], city) == tests[city]
```

Question 3b: Now, use the helper functions you wrote above to create a condensed data file for each city consisting only of the data fields indicated above. In the `/examples/` folder, you will

see an example datafile from the Bay Area Bike Share before and after conversion. Make sure that your output is formatted to be consistent with the example file.

```
In [6]: def condense_data(in_file, out_file, city):
            """
            This function takes full data from the specified input file
            and writes the condensed data to a specified output file. The city
            argument determines how the input file will be parsed.

            HINT: See the cell below to see how the arguments are structured!
            """

            with open(out_file, 'w') as f_out, open(in_file, 'r') as f_in:
                # set up csv DictWriter object - writer requires column names for the
                # first row as the "fieldnames" argument
                out_colnames = ['duration', 'month', 'hour', 'day_of_week', 'user_type']
                trip_writer = csv.DictWriter(f_out, fieldnames = out_colnames)
                trip_writer.writeheader()

                ## TODO: set up csv DictReader object ##
                trip_reader = csv.DictReader(f_in)

                # collect data from and process each row
                for row in trip_reader:
                    # set up a dictionary to hold the values for the cleaned and trimmed
                    # data point
                    new_point = {}

                    ## TODO: use the helper functions to get the cleaned data from  ##
                    ## the original data dictionaries.                              ##
                    ## Note that the keys for the new_point dictionary should match ##
                    ## the column names set in the DictWriter object above.         ##
                    new_point['duration'] = duration_in_mins(row, city)
                    new_point['month'], new_point['hour'], new_point['day_of_week'] = time_of_tr
                    new_point['user_type'] = type_of_user(row, city)

                    ## TODO: write the processed information to the output file.     ##
                    ## see https://docs.python.org/3/library/csv.html#writer-objects ##
                    trip_writer.writerow(new_point)


In [7]: # Run this cell to check your work
        city_info = {'Washington': {'in_file': './data/Washington-CapitalBikeshare-2016.csv',
                                    'out_file': './data/Washington-2016-Summary.csv'},
                     'Chicago': {'in_file': './data/Chicago-Divvy-2016.csv',
                                 'out_file': './data/Chicago-2016-Summary.csv'},
                     'NYC': {'in_file': './data/NYC-CitiBike-2016.csv',
                             'out_file': './data/NYC-2016-Summary.csv'}}
```

```
        for city, filenames in city_info.items():
            condense_data(filenames['in_file'], filenames['out_file'], city)
            print_first_point(filenames['out_file'])
```

```
City: Washington
OrderedDict([('duration', '7.123130912900001'),
             ('month', '3'),
             ('hour', '22'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Subscriber')])

City: Chicago
OrderedDict([('duration', '15.433333333333334'),
             ('month', '3'),
             ('hour', '23'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Subscriber')])

City: NYC
OrderedDict([('duration', '13.983333333333333'),
             ('month', '1'),
             ('hour', '0'),
             ('day_of_week', 'Friday'),
             ('user_type', 'Customer')])
```

> **Tip**: If you save a jupyter Notebook, the output from running code blocks will also be saved. However, the state of your workspace will be reset once a new session is started. Make sure that you run all of the necessary code blocks from your previous session to reestablish variables and functions before picking up where you left off.

Exploratory Data Analysis

Now that you have the data collected and wrangled, you're ready to start exploring the data. In this section you will write some code to compute descriptive statistics from the data. You will also be introduced to the `matplotlib` library to create some basic histograms of the data.

Statistics

First, let's compute some basic counts. The first cell below contains a function that uses the csv module to iterate through a provided data file, returning the number of trips made by subscribers and customers. The second cell runs this function on the example Bay Area data in the `/examples/` folder. Modify the cells to answer the question below.

Question 4a: Which city has the highest number of trips? Which city has the highest proportion of trips made by subscribers? Which city has the highest proportion of trips made by short-term customers?

Answer: The city which has the highest number of trips is NYC. The one that has the highest proportion of trips made by subscribers was Chicago and the city with highest proportion of trips made by short-term customers is NYC.

```
In [8]: def number_of_trips(filename):
            """
            This function reads in a file with trip data and reports the proportion of
            trips made by subscribers, customers, and total overall.
            """
            with open(filename, 'r') as f_in:
                # set up csv reader object
                reader = csv.DictReader(f_in)

                # initialize count variables
                n_subscribers = 0
                n_customers = 0

                # tally up ride types
                for row in reader:
                    if row['user_type'] == 'Subscriber':
                        n_subscribers += 1
                    else:
                        n_customers += 1

                # compute total number of rides
                n_total = n_subscribers + n_customers

                susbcribers_proportion = n_subscribers / n_total
                customers_proportion = n_customers / n_total
                # return tallies as a tuple
                return(susbcribers_proportion, customers_proportion, n_total)

In [9]: def highest_cities_values(summary_dict):
            """
            This function takes a dictionary with each city subscriber and
            customer proportion value and it's overall total and returns
            the highest city value in each value.
            """
            max_subscribers_prop = 0
            max_subscribers_city = None
            max_customers_prop = 0
            max_customers_city = None
            max_total_trips = 0
            max_total_trips_city = None

            for city, values in  summary_dict.items():
                if max_subscribers_prop < values[0]:
                    max_subscribers_prop = values[0]
                    max_subscribers_city = city
                if max_customers_prop < values[1]:
                    max_customers_prop = values[1]
                    max_customers_city = city
```

```
            if max_total_trips < values[2]:
                max_total_trips = values[2]
                max_total_trips_city = city

        return (max_subscribers_city, max_customers_city, max_total_trips_city)
```

In [16]:
```
## Modify this and the previous cell to answer Question 4a. Remember to run ##
## the function on the cleaned data files you created from Question 3.      ##

city_files = {
    'Washington': './data/Washington-2016-Summary.csv',
    'NYC': './data/NYC-2016-Summary.csv',
    'Chicago': './data/Chicago-2016-Summary.csv'
}

cities_value_dict = {}

for city, file in city_files.items():
    cities_value_dict[city] = number_of_trips(file)

cities_index = ["Subscribers", "Customers", "Total trips"]
highest_cities_index = ["Max subscribers city", "Max customers city", "Max total trips

df_cities = pd.DataFrame(cities_value_dict, index = cities_index)
s_highest_cities = pd.Series(highest_cities_values(cities_value_dict), index=highest_ci

print(df_cities)
print("\n")
print(s_highest_cities.to_string())
```

```
                 Chicago            NYC    Washington
Subscribers     0.762252       0.888359      0.780282
Customers       0.237748       0.111641      0.219718
Total trips  72131.000000  276798.000000  66326.000000


Max subscribers city        NYC
Max customers city      Chicago
Max total trips city        NYC
```

> **Tip**: In order to add additional cells to a notebook, you can use the "Insert Cell Above" and "Insert Cell Below" options from the menu bar above. There is also an icon in the toolbar for adding new cells, with additional icons for moving the cells up and down the document. By default, new cells are of the code type; you can also specify the cell type (e.g. Code or Markdown) of selected cells from the Cell menu or the dropdown in the toolbar.

Now, you will write your own code to continue investigating properties of the data.

Question 4b: Bike-share systems are designed for riders to take short trips. Most of the time, users are allowed to take trips of 30 minutes or less with no additional charges, with overage charges made for trips of longer than that duration. What is the average trip length for each city? What proportion of rides made in each city are longer than 30 minutes?

Answer: At following my responses for each city:

- Washington:
- Average trip length: 18.93 minutes
- rides duration above 30 minutes: 10.84%
- NYC:
- Average trip length: 15.81 minutes
- rides duration above 30 minutes: 7.32%
- Chicago:
- Average trip length: 16.56 minutes
- rides duration above 30 minutes: 8.35%

In [19]:
```
## Use this and additional cells to answer Question 4b.                 ##
##                                                                      ##
## HINT: The csv module reads in all of the data as strings, including  ##
## numeric values. You will need a function to convert the strings      ##
## into an appropriate numeric type before you aggregate data.          ##
## TIP: For the Bay Area example, the average trip length is 14 minutes ##
## and 3.5% of trips are longer than 30 minutes.                        ##
from csv import DictReader
def extract_trip_data(filename):
    with open(filename, 'r') as file_input:
        reader = DictReader(file_input)
        trips_collection = []
        for row in reader:
            dict_trip = {}
            dict_trip['duration'] = float(row['duration'])
            dict_trip['month'] = int(row['month'])
            dict_trip['hour'] = int(row['hour'])
            dict_trip['day_of_week'] = row['day_of_week']
            dict_trip['user_type'] = row['user_type']
            trips_collection.append(dict_trip)
        return trips_collection
```

In [20]:
```
city_dict = {
    'Washington': './data/Washington-2016-Summary.csv',
    'NYC': './data/NYC-2016-Summary.csv',
    'Chicago': './data/Chicago-2016-Summary.csv'
}

washington_data = extract_trip_data(city_dict['Washington'])
nyc_data = extract_trip_data(city_dict['NYC'])
chicago_data = extract_trip_data(city_dict['Chicago'])
```

In [22]:
```
## What is the average trip length for each city?
##What proportion of rides made in each city are longer than 30 minutes?
```

```
        def get_trip_data_promedy_and_proportion(trip_data):
            duration_promedy = 0
            trips_above_30_mins_promedy = 0
            for row in trip_data:
                duration_promedy += row['duration']
                if row['duration'] >= 30:
                    trips_above_30_mins_promedy += 1
            duration_promedy /= len(trip_data)
            trips_above_30_mins_promedy = (trips_above_30_mins_promedy / len(trip_data)) * 100

            return duration_promedy, trips_above_30_mins_promedy

        cities_trip_prom = {
            'Washington': get_trip_data_promedy_and_proportion(washington_data),
            'NYC': get_trip_data_promedy_and_proportion(nyc_data),
            'Chicago': get_trip_data_promedy_and_proportion(chicago_data),
        }

        cities_trip_prom_index = ["Average Promedy", "Trips above 30 min proportion"]

        df_cities_prip_prom = pd.DataFrame(data= cities_trip_prom, index=cities_trip_prom_index

        print(df_cities_prip_prom)
```

```
                              Chicago        NYC  Washington
Average Promedy             16.563629  15.812593   18.932911
Trips above 30 min proportion  8.347313   7.316888   10.838887
```

Question 4c: Dig deeper into the question of trip duration based on ridership. Choose one city. Within that city, which type of user takes longer rides on average: Subscribers or Customers?

Answer: Taking Chicago city as reference, 'Customer' users are which takes longer rides on average.

```
In [25]: ## Use this and additional cells to answer Question 4c. If you have    ##
         ## not done so yet, consider revising some of your previous code to    ##
         ## make use of functions for reusability.                              ##
         ##                                                                     ##
         ## TIP: For the Bay Area example data, you should find the average     ##
         ## Subscriber trip duration to be 9.5 minutes and the average Customer ##
         ## trip duration to be 54.6 minutes. Do the other cities have this     ##
         ## level of difference?                                                ##
         def get_usertypes_promedy(trip_data):
             """
             Gets a trips data dictionary and returns the user type
             duration trip promedies.
             """
```

```
            subs_trip_prom = 0
            subs_trip_qty = 0
            cust_trip_prom = 0
            cust_trip_qty = 0

            for row in trip_data:
                if row['user_type'] == 'Subscriber':
                    subs_trip_qty += 1
                    subs_trip_prom += row['duration']
                if row['user_type'] == 'Customer':
                    cust_trip_qty += 1
                    cust_trip_prom += row['duration']

            subs_trip_prom /= subs_trip_qty
            cust_trip_prom /= cust_trip_qty

            return subs_trip_prom, cust_trip_prom
```

```
In [35]: chicago_filedata = './data/Chicago-2016-Summary.csv'
         chicago_data = extract_trip_data(chicago_filedata)
         chicago_promedies = list(get_usertypes_promedy(chicago_data))
         chicago_prom_columns = ["Chicago average trip duration"]
         chicago_prom_index = ["by Subscribers", "by Customers"]

         df_chicago_prom = pd.DataFrame(data=chicago_promedies, index=chicago_prom_index, column
         print(df_chicago_prom)
```

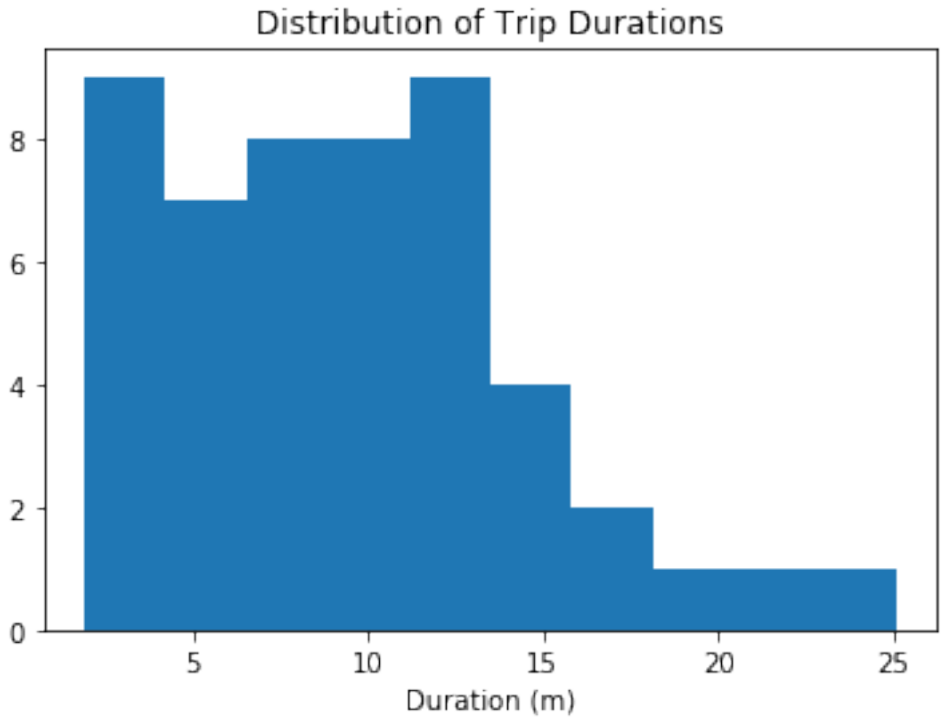
```
                Chicago average trip duration
by Subscribers                      12.067202
by Customers                        30.979781
```

Visualizations

The last set of values that you computed should have pulled up an interesting result. While the mean trip time for Subscribers is well under 30 minutes, the mean trip time for Customers is actually *above* 30 minutes! It will be interesting for us to look at how the trip times are distributed. In order to do this, a new library will be introduced here, `matplotlib`. Run the cell below to load the library and to generate an example plot.

```
In [8]: # load library
        import matplotlib.pyplot as plt

        # this is a 'magic word' that allows for plots to be displayed
        # inline with the notebook. If you want to know more, see:
        # http://ipython.readthedocs.io/en/stable/interactive/magics.html
        %matplotlib inline

        # example histogram, data taken from bay area sample
        data = [ 7.65,  8.92,  7.42,  5.50, 16.17,  4.20,  8.98,  9.62, 11.48, 14.33,
```

```
        19.02, 21.53,  3.90,  7.97,  2.62,  2.67,  3.08, 14.40, 12.90,  7.83,
        25.12,  8.30,  4.93, 12.43, 10.60,  6.17, 10.88,  4.78, 15.15,  3.53,
         9.43, 13.32, 11.72,  9.85,  5.22, 15.10,  3.95,  3.17,  8.78,  1.88,
         4.55, 12.68, 12.38,  9.78,  7.63,  6.45, 17.38, 11.90, 11.52,  8.63,]
plt.hist(data)
plt.title('Distribution of Trip Durations')
plt.xlabel('Duration (m)')
plt.show()
```



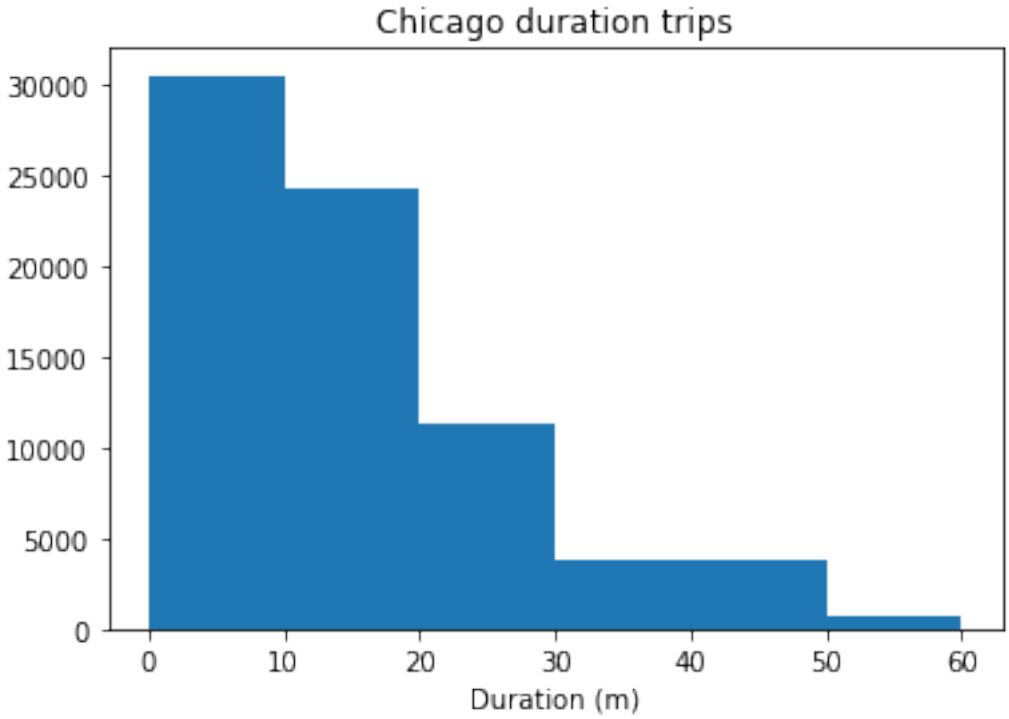
In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. The `.title()` and `.xlabel()` functions provide some labeling for plot context.

You will now use these functions to create a histogram of the trip times for the city you selected in question 4c. Don't separate the Subscribers and Customers for now: just collect all of the trip times and plot them.

```
In [8]: ## Use this and additional cells to collect all of the trip times as a list ##
        ## and then use pyplot functions to generate a histogram of trip times.     ##
        chicago_trips_duration = []
        for row in chicago_data:
            chicago_trips_duration.append(round(row['duration'], 2))
```

```
In [9]: from matplotlib import pyplot
```

```
%matplotlib inline

pyplot.hist(chicago_trips_duration, bins=[0, 10, 20, 30, 50, 60])
pyplot.title("Chicago duration trips")
pyplot.xlabel("Duration (m)")
pyplot.show()
```



If you followed the use of the `.hist()` and `.show()` functions exactly like in the example, you're probably looking at a plot that's completely unexpected. The plot consists of one extremely tall bar on the left, maybe a very short second bar, and a whole lot of empty space in the center and right. Take a look at the duration values on the x-axis. This suggests that there are some highly infrequent outliers in the data. Instead of reprocessing the data, you will use additional parameters with the `.hist()` function to limit the range of data that is plotted. Documentation for the function can be found [here].

Question 5: Use the parameters of the `.hist()` function to plot the distribution of trip times for the Subscribers in your selected city. Do the same thing for only the Customers. Add limits to the plots so that only trips of duration less than 75 minutes are plotted. As a bonus, set the plots up so that bars are in five-minute wide intervals. For each group, where is the peak of each distribution? How would you describe the shape of each distribution?

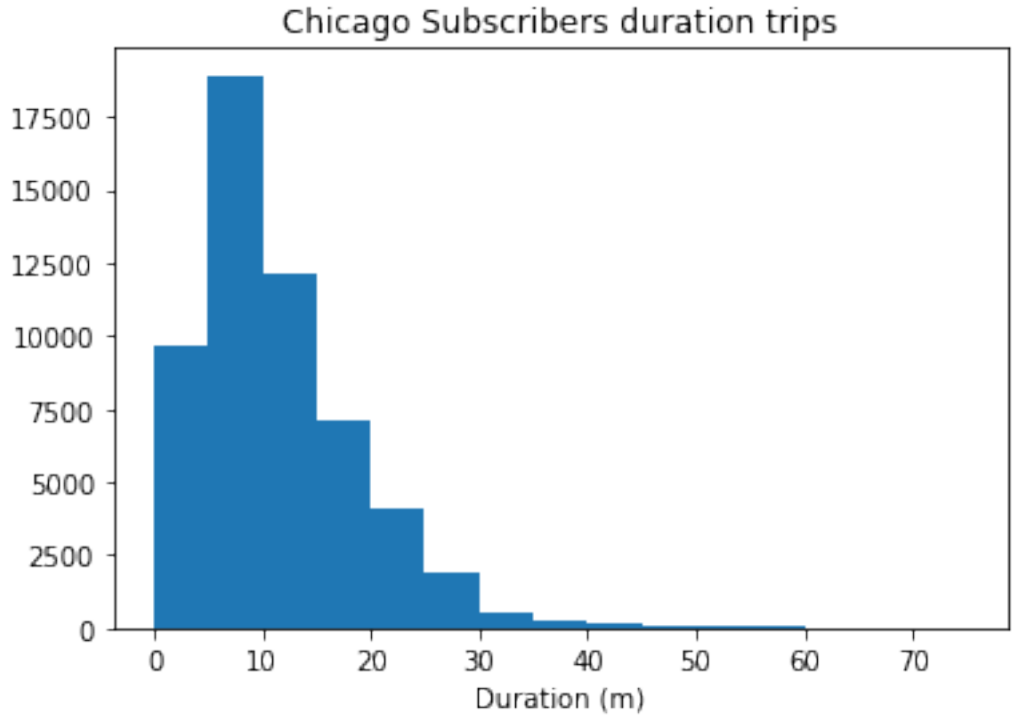
Answer: The peak of each distribution are: - 0 - 5 : 10000 - 5 - 10 : 17500+ - 10 - 15: 12500 - 15 - 20: 7500 - 20 - 25: 5000 - 25 - 75: 2500-

The distribution is right skewed.

```
In [10]: ## Use this and additional cells to answer Question 5. ##
         chicago_subscribers_duration = []
```

```
        for row in chicago_data:
            if row['user_type'] == 'Subscriber':
                chicago_subscribers_duration.append(round(row['duration'], 2))
```

```
In [11]: %matplotlib inline

         bin_edges = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75]

         pyplot.hist(chicago_subscribers_duration, bins=bin_edges)
         pyplot.title("Chicago Subscribers duration trips")
         pyplot.xlabel("Duration (m)")
         pyplot.show()
```



Performing Your Own Analysis

So far, you've performed an initial exploration into the data available. You have compared the relative volume of trips made between three U.S. cities and the ratio of trips made by Subscribers and Customers. For one of these cities, you have investigated differences between Subscribers and Customers in terms of how long a typical trip lasts. Now it is your turn to continue the exploration in a direction that you choose. Here are a few suggestions for questions to explore:

- How does ridership differ by month or season? Which month / season has the highest ridership? Does the ratio of Subscriber trips to Customer trips change depending on the month or season?
- Is the pattern of ridership different on the weekends versus weekdays? On what days are Subscribers most likely to use the system? What about Customers? Does the average duration of rides change depending on the day of the week?

- During what time of day is the system used the most? Is there a difference in usage patterns for Subscribers and Customers?

If any of the questions you posed in your answer to question 1 align with the bullet points above, this is a good opportunity to investigate one of them. As part of your investigation, you will need to create a visualization. If you want to create something other than a histogram, then you might want to consult the Pyplot documentation. In particular, if you are plotting values across a categorical variable (e.g. city, user type), a bar chart will be useful. The documentation page for `.bar()` includes links at the bottom of the page with examples for you to build off of for your own use.

Question 6: Continue the investigation by exploring another question that could be answered by the data available. Document the question you want to explore below. Your investigation should involve at least two variables and should compare at least two groups. You should also use at least one visualization as part of your explorations.

Answer: My investigation is focused in Chicago City based in two questions:

- Which are the hours that service is used the most on weekends and weekdays?
- Which hours the Subscribers and Customers uses the service the most on weekends and weekdays?

In [79]:
```
"""
This script is used to get all necesary data to perform the
visualizations of the information needed to answer the questions
asked before.
I discriminated the info in the following pattern:

Subscriber
     |-> Weekday Trips info
     |-> Weekend Trips info
Customer
     |-> Weekday Trips info
     |-> Weekend Trips info

General Weekday trips info =
    Subscriber Weekday Trips info + Customer Weekday Trips info

General Weekend trips info =
    Subscriber Weekend Trips info + Customer Weekend Trips info
"""
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']


subscriber_weekday_trips = []
customer_weekday_trips = []
subscriber_weekend_trips = []
customer_weekend_trips = []

for row in chicago_data:
```

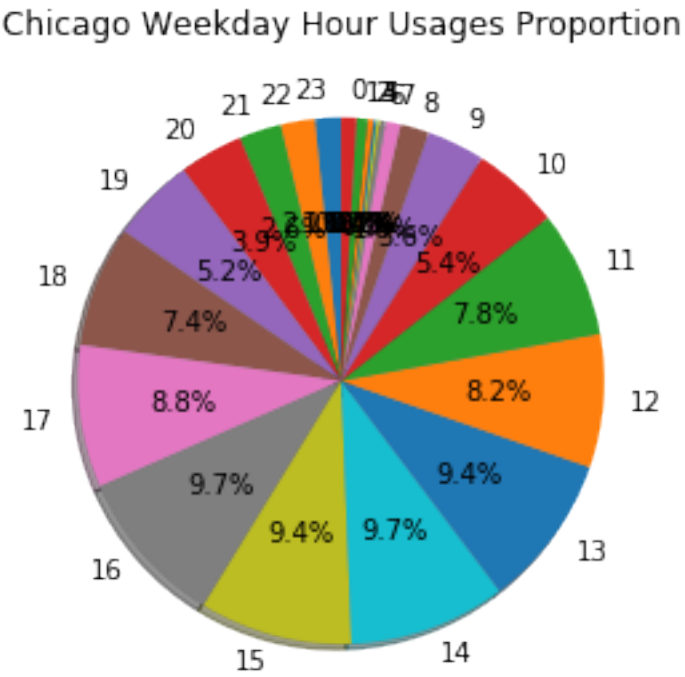
```
        if row['day_of_week'] in weekdays:
            if row['user_type'] == 'Subscriber':
                subscriber_weekday_trips.append(row)
            else:
                customer_weekday_trips.append(row)
        else:
            if row['user_type'] == 'Subscriber':
                subscriber_weekend_trips.append(row)
            else:
                customer_weekend_trips.append(row)
```

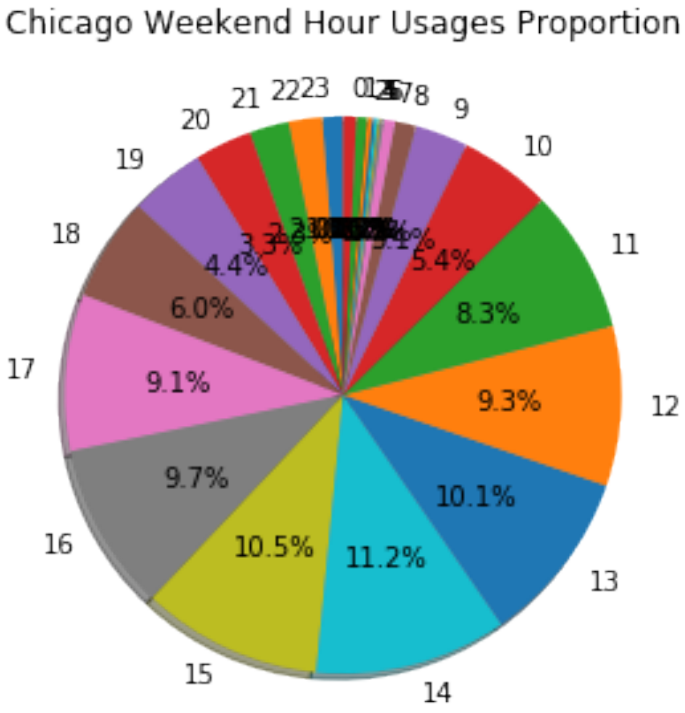
In [82]:
```
from collections import Counter

subscriber_weekday_hours = []
subscriber_weekend_hours = []
customer_weekday_hours = []
customer_weekend_hours = []

for row in subscriber_weekday_trips:
    subscriber_weekday_hours.append(row['hour'])

for row in subscriber_weekend_trips:
    subscriber_weekend_hours.append(row['hour'])

for row in customer_weekday_trips:
    customer_weekday_hours.append(row['hour'])

for row in customer_weekend_trips:
    customer_weekend_hours.append(row['hour'])

subscriber_weekday_hours = sorted(subscriber_weekday_hours, reverse=True)
subscriber_weekend_hours = sorted(subscriber_weekend_hours, reverse=True)
customer_weekday_hours = sorted(customer_weekday_hours, reverse=True)
customer_weekend_hours = sorted(customer_weekend_hours, reverse=True)

subscriber_weekday_sum_hours = dict(Counter(subscriber_weekday_hours))
subscriber_weekend_sum_hours = dict(Counter(subscriber_weekend_hours))
customer_weekday_sum_hours = dict(Counter(customer_weekday_hours))
customer_weekend_sum_hours = dict(Counter(customer_weekend_hours))

general_weekday_sum_hours = {}
general_weekday_sum_hours.update(subscriber_weekday_sum_hours)
general_weekday_sum_hours.update(customer_weekday_sum_hours)


general_weekend_sum_hours = {}
general_weekend_sum_hours.update(subscriber_weekend_sum_hours)
general_weekend_sum_hours.update(customer_weekend_sum_hours)
```

Which are the hours that service is used the most on weekends and weekdays? Answer: The hours that service is used the most are:

- Weekdays: 2pm and 4pm with each 9.7 percent of trips done.
- Weekends: 2pm with a 11.2 percent of trips done.

```
In [90]: %matplotlib inline
         import matplotlib.pyplot as plt

         weekday_labels = list(general_weekday_sum_hours.keys())
         weekday_data = list(general_weekday_sum_hours.values())

         fig1, ax1 = plt.subplots()
         ax1.pie(weekday_data, labels=weekday_labels, autopct='%1.1f%%', shadow=True, startangle
         ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
         plt.title('Chicago Weekday Hour Usages Proportion\n')

         weekend_labels = list(general_weekend_sum_hours.keys())
         weekend_data = list(general_weekend_sum_hours.values())

         fig2, ax2 = plt.subplots()
         ax2.pie(weekend_data, labels=weekend_labels, autopct='%1.1f%%', shadow=True, startangle
         ax2.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
         plt.title('Chicago Weekend Hour Usages Proportion\n')

         plt.tight_layout()
         plt.show()
```



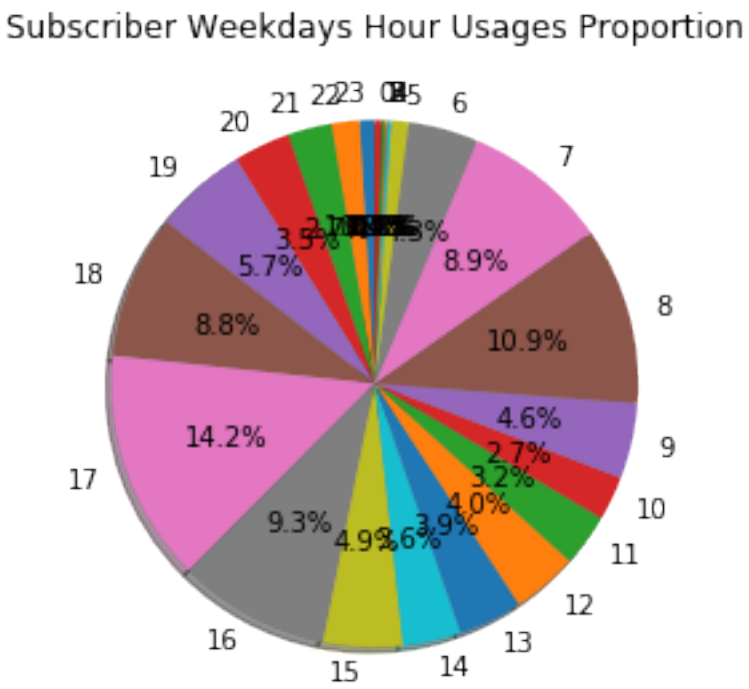


Which hours the Subscribers and Customers uses the service the most on weekends and weekdays? **Answer**: The hours that service is used the most by user types are:

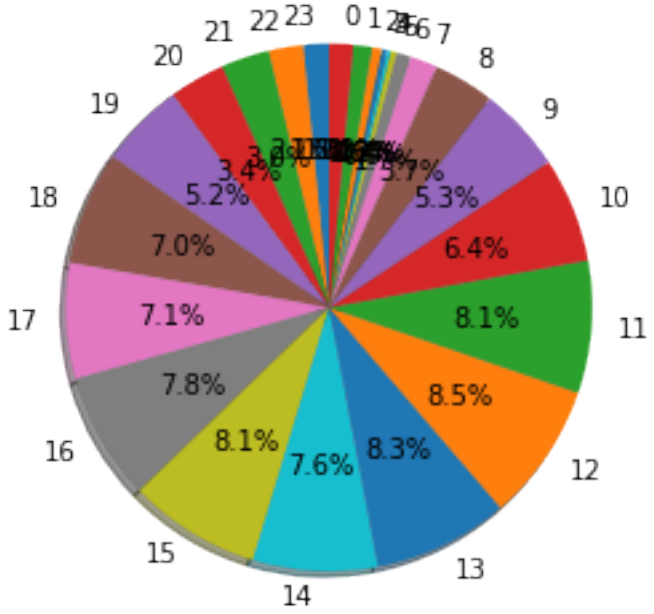
- Subscribers:
- Weekdays: 5pm with a 14.2 percent of trips done.
- Weekends: 12pm with a 8.5 percent of trips done.
- Customers:
- Weekdays: 2pm and 4pm with each a 9.7 percent of trips done.
- Weekends: 4pm with a 11.2 percent of trips done.

```
In [96]: %matplotlib inline
         import matplotlib.pyplot as plt

         sub_weekday_labels = list(subscriber_weekday_sum_hours.keys())
         sub_weekday_data = list(subscriber_weekday_sum_hours.values())
         sub_weekend_labels = list(subscriber_weekend_sum_hours.keys())
         sub_weekend_data = list(subscriber_weekend_sum_hours.values())

         cust_weekday_labels = list(customer_weekday_sum_hours.keys())
         cust_weekday_data = list(customer_weekday_sum_hours.values())
         cust_weekend_labels = list(customer_weekend_sum_hours.keys())
```

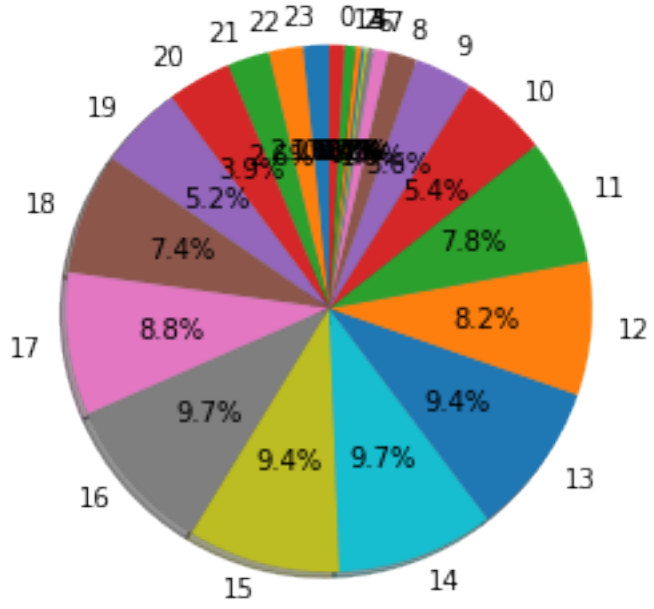
```
cust_weekend_data = list(customer_weekend_sum_hours.values())

fig1, ax1 = plt.subplots()
ax1.pie(sub_weekday_data, labels=sub_weekday_labels, autopct='%1.1f%%', shadow=True, st
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('Subscriber Weekdays Hour Usages Proportion\n')

fig2, ax2 = plt.subplots()
ax2.pie(sub_weekend_data, labels=sub_weekend_labels, autopct='%1.1f%%', shadow=True, st
ax2.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('Subscriber Weekend Hour Usages Proportion\n')

fig3, ax3 = plt.subplots()
ax3.pie(cust_weekday_data, labels=cust_weekday_labels, autopct='%1.1f%%', shadow=True,
ax3.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('Customer Weekdays Hour Usages Proportion\n')

fig4, ax4 = plt.subplots()
ax4.pie(cust_weekend_data, labels=cust_weekend_labels, autopct='%1.1f%%', shadow=True,
ax4.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('Customer Weekend Hour Usages Proportion\n')

plt.tight_layout()
plt.show()
```

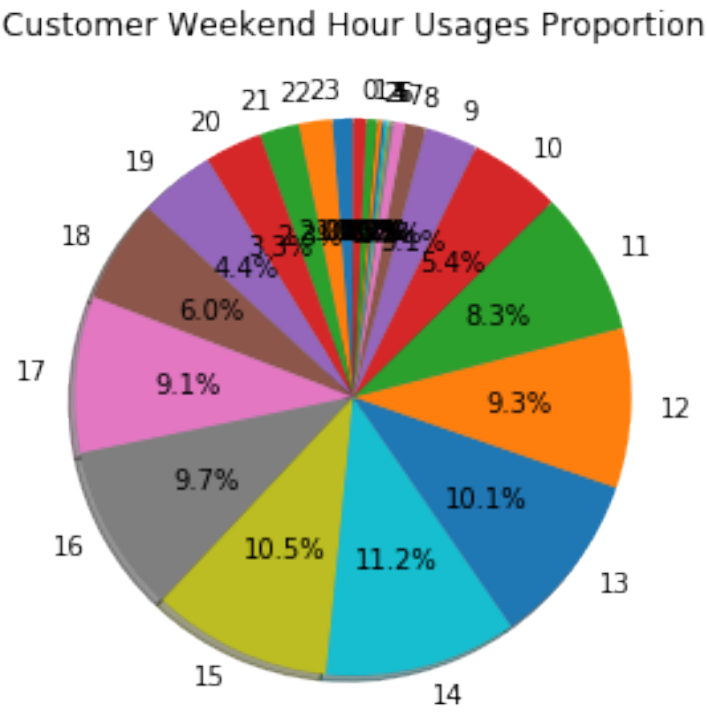


Subscriber Weekend Hour Usages Proportion



Customer Weekdays Hour Usages Proportion





Conclusions

Congratulations on completing the project! This is only a sampling of the data analysis process: from generating questions, wrangling the data, and to exploring the data. Normally, at this point in the data analysis process, you might want to draw conclusions about the data by performing a statistical test or fitting the data to a model for making predictions. There are also a lot of potential analyses that could be performed on the data which are not possible with only the data provided. For example, detailed location data has not been investigated. Where are the most commonly used docks? What are the most common routes? As another example, weather has potential to have a large impact on daily ridership. How much is ridership impacted when there is rain or snow? Are subscribers or customers affected more by changes in weather?

Question 7: Putting the bike share data aside, think of a topic or field of interest where you would like to be able to apply the techniques of data science. What would you like to be able to learn from your chosen subject?

Answer: I would like apply techniques of data science into more AI related topics like human-application interactions and robotics. I'm not sure if there is a topic that could be fit in the topics of this course(regarding about my choosed topic before), although i think if i need to pick a topic that could be able to learn i would be correct classification of the wrangled data, as it's used in the machine learning area.

> **Tip**: If we want to share the results of our analysis with others, we aren't limited to giving them a copy of the jupyter Notebook (.ipynb) file. We can also export the Notebook output in a form that can be opened even for those without Python installed. From the **File** menu in the upper left, go to the **Download as** submenu. You can then choose a different format that can be viewed more generally, such as HTML (.html) or PDF (.pdf). You may need additional packages or software to perform these exports.

```
In [97]: import nbconvert
```